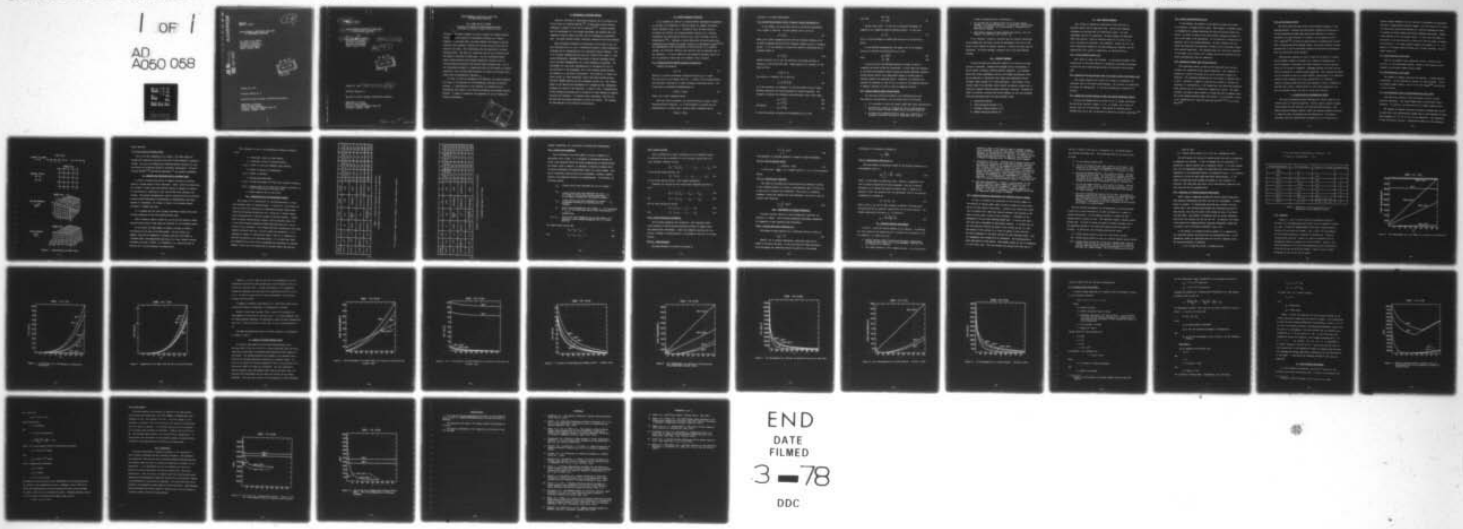
AD-A050 058 ARIZONA UNIV TUCSON DEPT OF AEROSPACE AND MECHANICA--ETC F/G 12/1
DIRECT NUMERICAL SOLUTION OF LARGE SETS OF SIMULTANEOUS EQUATIO--ETC(U)
JAN 78 H A KAMEL, M W MCCABE N00014-75-C-0837

UNCLASSIFIED TR-3 NL

1 OF 1
AD A050 058



END
DATE FILMED
3-78
DDC

12

N00014-75-C-0837

DIRECT NUMERICAL SOLUTION OF LARGE SETS OF SIMULTANEOUS EQUATIONS

H.A. Kamel & M.W. McCabe
University of Arizona
Aerospace and Mechanical
Engineering Department
Tucson, Arizona 85721

January 20, 1978

Technical Report No. 3

Approved for public release, distribution unlimited

Department of the Navy
Office of Naval Research
Structural Mechanics Program (Code 474)
Arlington, Virginia 22217

(15) N00014-75-C-0837

(6) DIRECT NUMERICAL SOLUTION OF LARGE SETS OF SIMULTANEOUS EQUATIONS.

(10) H.A. Kamel and M.W. McCabe
University of Arizona
Aerospace and Mechanical
Engineering Department
Tucson, Arizona 85721

(9) Technical rept.

(14) TR-3

January 20, 1978 (11) 20 Jan 78 (12) 47p.

Technical Report No. 3

Approved for public release, distribution unlimited

Department of the Navy
Office of Naval Research
Structural Mechanics Program (Code 474)
Arlington, Virginia 22217

402 192

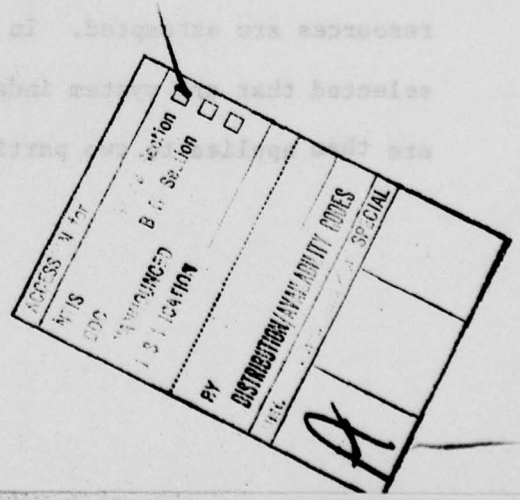
DIRECT NUMERICAL SOLUTION OF LARGE SETS OF SIMULTANEOUS EQUATIONS

H.A. KAMEL AND M.W. MCCABE

Aerospace and Mechanical Engineering Department
University of Arizona, Tucson, AZ 85721, U.S.A.

Abstract -- The paper attempts to survey, classify and compare methods availab[illegible]r the solution of simultaneous equations on a digital computer. It is known that, for direct methods, the amount of central processor time required to perform the solution varies little from one method to the other. More interesting than the algorithm is the data handling method. The basis for comparison here is core utilization and the number of required I/O operations. Low core utilization means higher program capacity and increased generality, whereas a low I/O activity improves efficiency and reduces system residence time. One of the interesting findings in a comparative study of the demands on computer resources is that the data handling method is indeed the deciding factor, rather than the mathematical algorithm.

In order to avoid too mathematical an approach, the paper recognizes the origin of the equations in the mathematical modelling of physical problems. A classification of such problems is attempted and its implications with regard to the solution procedures are assessed whenever possible. A number of parameters are suggested for use in program performance measurements.



I. EVALUATION OF SOLUTION METHODS

Numerical solutions of simultaneous equations may be performed with either direct or iterative methods.[1] In some cases, hybrid methods, combining an element of both, may be employed.[2,3] Iterative methods have the advantages of a low storage requirement and possible fast convergence in certain cases, but their rate of convergence is unpredictable. The paper concerns itself, therefore, primarily with direct methods.

Many misleading statements and conclusions can be found in the current literature due to a lack of distinction between a basic mathematical technique and its implementation on an electronic digital computer. Just as an excellent mathematician may have his "tricks of the trade," so does a good professional programmer have access to special knowledge resulting in efficient implementation of a basic mathematical algorithm. The performance of a particular implementation is influenced not only by the excellence of the programmer, but by the available system software and hardware at a particular installation. The presence of a large core, direct access (or index sequential) files, high speed backing storage, efficient FORTRAN compiler and file management system, etc., has a great impact on the capacity and performance of a program and will undoubtedly influence the choice of the algorithm. In view of this, a classification of data handling techniques and an evaluation of their demands on computer resources are attempted. In comparing these algorithms, criterion are selected that are system independent, as much as possible. The findings are then applied to two particular installations.

II. DIRECT METHODS OF SOLUTION

It is customary to refer to a solution method (mathematical algorithm) by the name of an originator to whom the method is commonly attributed (Gauss, Cholesky, Crout, etc.). Instead of this, two main solution strategies are defined, the triangularization methods (e.g., Gaussian elimination) and factorization methods (e.g., Crout, Cholesky, Inverse decomposition [4]). Although it can be shown that the factorization and triangularization methods are closely related in a mathematical sense[1,5,6], the computational steps and sequence of data retrieval, in a computer program, are distinctly different and therefore play an important part in any comparison. It can be shown that the greatest difference relates to the I/O operations rather than the arithmetic effort required.

II.1 *Triangularization Method (Gaussian Elimination)*

Consider the equation

$$\underline{K}\,\underline{r} = \underline{R} \tag{1}$$

where $\underline{K}$ is a square non-singular coefficient matrix, $\underline{R}$ is a right-hand side and $\underline{r}$ is an unknown vector(s). A series of row transformations are performed, resulting in an upper triangular coefficient matrix. The process may be represented mathematically as

$$(\underline{L}\,\underline{K})\,\underline{r} = (\underline{L}\,\underline{R}) \tag{2}$$

where $\underline{L}$ is a lower triangular matrix.

Once this form is achieved, the solution proceeds through a simple back-substitution operation. It is also possible to perform the triangularization in reverse order, using an upper triangular matrix,

$$(\underline{U}\,\underline{K})\,\underline{r} = (\underline{U}\,\underline{R}) \tag{2a}$$

followed by a forward substitution.

II.2 *Factorization Method (Crout, Cholesky, Inverse decomposition)*

In this method, the coefficient matrix is factorized (decomposed) into a number of matrices. The most general case is given by

$$\underline{K} = \underline{L}\,\underline{d}\,\underline{U} \tag{3}$$

where $\underline{L}$ is a lower triangular matrix with unit diagonal elements, $\underline{U}$ is an upper triangular matrix with unit diagonal elements and $\underline{d}$ is a diagonal matrix. It is also possible to reverse the scheme by expressing the coefficient matrix as

$$\underline{K} = \underline{U}\,\underline{d}\,\underline{L}. \tag{3a}$$

Whether equations (3) or (3a) are utilized, the program proceeds to compute $\underline{L}$, $\underline{d}$ and $\underline{U}$ and store them. Using equation (3), equation (1) now takes the form

$$\underline{L}\,\underline{d}\,\underline{U}\,\underline{r} = \underline{R}. \tag{4}$$

The solution to equation (4) is given by:

$$\underline{r} = \underline{U}^{-1}\underline{d}^{-1}\underline{L}^{-1}\underline{R}. \tag{5}$$

It is not necessary, or advisable, to form the inverse of any of these matrices explicitly since matrix sparsity is thereby destroyed. One proceeds to evaluate the following vectors directly, and in this order,

$$\underline{r}_1 = \underline{L}^{-1}\underline{R} \tag{5a}$$

$$\underline{r}_2 = \underline{d}^{-1}\underline{r}_1 \tag{5b}$$

and finally

$$\underline{r}_3 = \underline{U}^{-1}\underline{r}_2. \tag{5c}$$

In the Crout method, the matrix $\underline{U}$ is multiplied by $\underline{d}$ to form

$$\underline{U}^* = \underline{d}\,\underline{U}$$

such that

$$\underline{K} = \underline{L}\,\underline{U}^*. \tag{6}$$

Special cases arise. In the case of structural problems, for example, $\underline{K}$ is a symmetric positive definite matrix. In this case,

$$\underline{L} = \underline{U}^t. \tag{7}$$

And it is only necessary to compute and store one of the matrices $\underline{L}$ and $\underline{U}$.

In the Cholesky decomposition, the square root of the diagonal matrix $\underline{d}$ is obtained and multiplied by $\underline{L}$ so that

$$\underline{K} = \underline{L}^{*t}\underline{L}^* \tag{8}$$

where

$$\underline{L}^* = \underline{d}^{\frac{1}{2}}\underline{L} \tag{8a}$$

In both the Crout and Cholesky methods an attempt is made to eliminate the matrix $\underline{d}$ from the formulation. In both cases the variation on the basic factorization scheme is minor and causes additional problems. Neither method results in an appreciable saving of storage space or seriously affects the computing effort. Therefore, the original formulation of equations (4) and (5) will be used for non-symmetric matrices. In additon, equation (7) will be used for symmetric matrices.

II.3 *General Remarks About Direct Methods*

For both basic solution procedures, the triangularization and factorization (decomposition), one can safely state the following:

a. It is possible to solve for several right-hand sides simultaneously.

b. No additional storage is required for the $\underline{L}$, $\underline{U}$ and $\underline{d}$ matrices. They can occupy the same storage which $\underline{K}$ occupied originally.

c. By saving the triangular matrices ($\underline{L}$ $\underline{K}$), $\underline{d}$, $\underline{L}$, and/or $\underline{U}$, it is possible to solve for new sets of right-hand sides without

further triangularization or factorization.

d. Both methods may be combined with any of the data storage schemes, to be described under IV. In particular, both methods, and their derivatives, can be generalized to the hyper-row or hyper-matrix schemes.

e. Both methods require the same computational effort. See, for example, the operation count in ref. [4].

It is, therefore, natural to conclude that the relative superiority of one scheme over the other can only be determined, if at all, on the basis of their demand on computer resources -- mainly core space and I/O operations. The choice becomes, therefore, one of the data handling strategy.

III. ITERATIVE METHODS

Iterative methods are potentially useful in the solution of large systems of equations. Although few programs utilize such methods, it may be unwise to dismiss iterative methods altogether. Such methods still hold certain advantages, such as low storage requirements, which may be important in conjunction with large three dimensional solids problems, where storage is of prime importance -- particularly if the current trend of increasingly faster central processors, coupled with a relatively stagnant backing storage technology, continues. Although the paper is primarily concerned with direct methods, we list the following iterative approaches which are most prominent today:

a. Gauss-Seidel Method

b. Block Relaxation Methods [2,3]

c. Conjugate Gradient Method [7,8]

d. Dynamic Relaxation Method [9]

IV. DATA HANDLING METHODS

Core storage is usually not sufficient to hold both the coefficient matrix and the right-hand side. Special data handling techniques are utilized with two objectives in mind: low core requirement and few I/O operations. The main schemes are described under this section. They are labelled using I for in core, R for row, G for group or partition, S for submatrix. Except for IB, a two letter designation stands for the disk storage unit (record), and the computational unit (pivot), respectively, as described below.

IV.1 *In-Core Banded Storage (IB)*

Only useful in small size problems. In non-linear problems, where problem size is often kept down out of necessity, and where one attempts to avoid the use of mass storage devices, the scheme is still heavily used.

IV.2 *Banded Out-of-Core Storage, Rows (or Columns) Stored Individually (RR)*

This scheme requires only enough core storage to accommodate two individual rows of the matrix simultaneously. The solution is accomplished via single row combinations. It has the disadvantage of excessive I/O activity.

IV.3 *Banded Out-of-Core Storage, by Row (or Column) Partitions (GR,GG)*

In which the banded matrix is stored in row, or column, partitions. Each partition contains a number of rows, or columns, and is read, or written, with one I/O instruction. This method of operation may be combined with row by row, or partition by partition, solution algorithms.[10]

IV.4 *Sparse Representation [11,12]*

In this method, the sparsity of the matrix is taken into account on an element, or element string, basis. Each element, or element string, is accompanied by a header describing its size and position within the matrix. An effort is made so that only "active" strings are present in core at any stage of the computation. It is difficult to evalute such an approach, due to its problem dependency. We feel, however, that the method can essentially be described in terms of one of the other schemes, with variable band-width. The problems chosen as a basis for comparison in this paper do not reflect this property, and we can therefore exclude sparse representation from our comparison.

IV.5 *Hypermatrix Method (SG) [13,14,15,16,17]*

The coefficient matrix, as well as the right-hand side, are partitioned into compatible blocks called submatrices. Each submatrix is treated as a record to be moved in or out of core with a single I/O operation. The individual submatrices are not stored, or operated upon, if zero. Therefore, a number of pointers are utilized to indicate the existence, or non-existence, of the submatrices, and their disk addresses. These pointers may be, in themselves, regarded as a matrix. For large problems, this matrix again presents a problem and must be handled using one of the schemes described under this section. It is possible to treat it as a banded matrix, a sparsely populated matrix[12,14] or as a hypermatrix[15].

IV.6 *Miscellaneous Methods*

The above cases have been chosen as well-defined examples of data handling methods. Instances may exist where programs use variations on the above methods that make them perhaps more efficient in certain situations, but also difficult to classify. As an example, the Wav- Front technique[18] utilizes the sparse method of representation coupled with a sophisticated disk handling scheme. The method combines a banded approach with a form of band-width optimization performed simultaneously with the coefficient matrix assembly.

Buffering may sometimes be useful, where an I/O buffer is created, which contains a number of records (row, columns or hypermatrices). If an I/O operation is initiated, the buffer directory is first consulted before an I/O access is attempted. Exchanges between the buffer and the disk are governed by a suitable paging algorithm. This technique increases the I/O overhead but may be of great benefit, particularly if an additional amount of fast secondary core (e.g. Extended Core) is available. A similar situation arises with some mini-computers in which a FORTRAN program may only access a limited amount of core during computation, but may use addresses beyond its area for block data transfers.

V. CLASSIFICATION OF MATHEMATICAL MODELS

The type of mathematical model employed has a direct impact on the correct choice of solution algorithm. Ideally, general purpose programs should employ the most efficient all round algorithm possible. Many special purpose programs, however, employ the simplest possible algorithm to solve the class of problems they are designed for. We believe, therefore, that the classification attempted here is of value both to

special program designers, and as a measure of performance of algorithms employed in large general purpose programs. For the purpose of our study, we choose some typical grids employed in a discrete mathematical model, as typified by finite element and finite difference approaches. Figure 1 shows sketches of the four models chosen. In each case, the number of grid nodes in any one direction is given by N, and the number of degrees of freedom by f. In a heat transfer problem, for example, f = 1. It is equal to 3 in elastic solids problems and 6 in shell problems. The four models chosen are:

V.1 *One-Dimensional Grids (E1S1)*

This is the simplest case, admittedly trivial, utilized in the solution of a one-dimensional problem. Models of this nature are best handled in core using a banded storage scheme.

V.2 *Two-Dimensional Grids (E2S2)*

The demands on computer resources are moderate. A banded approach is effective. In-core handling is possible for small problems. Out-of-core banded schemes (GR,GG), such as in the SAP program[10] perform well in this situation.

V.3 *Two-Dimensional Grids in Three-Dimensional Space (E2S3)*

Such models are typical of shell structures, and are of great practical importance. The chosen example grid is sufficiently representative. In practice, such grids may be of a highly complex nature.

The demands on computer resources are greatly increased and, as will be shown later, more sophisticated schemes such as those employed in large scale programs [11, 13, 14, 15, 16, 17] are imperative. The complexity of node interaction dictates a sophisticated method for the handling of

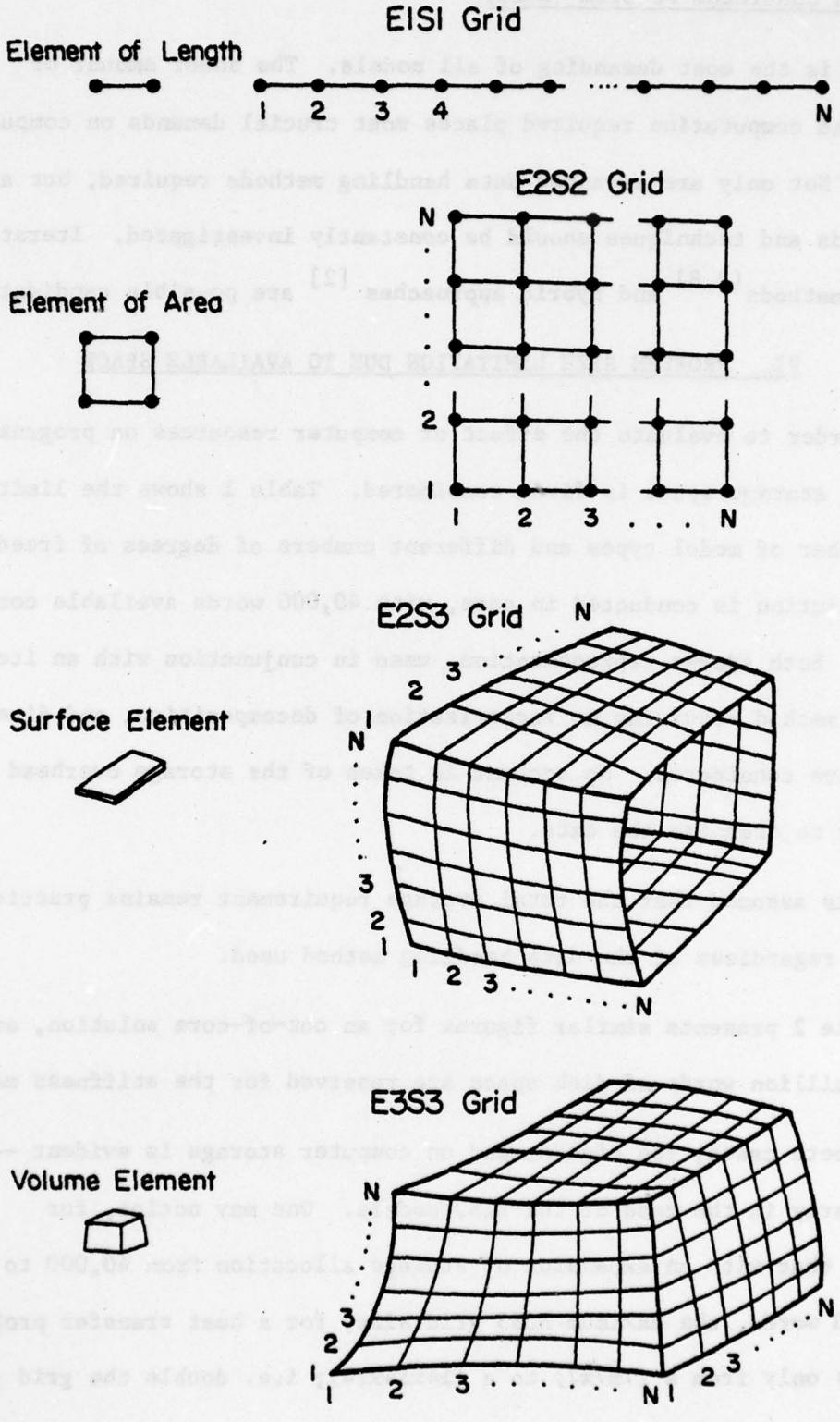


Figure 1. Classification of Model Grids

matrix sparsity.

V.4 *Solid Continuum Problem (E3S3)*

This is the most demanding of all models. The sheer amount of storage and computation required places most crucial demands on computing systems. Not only are advanced data handling methods required, but also new methods and techniques should be constantly investigated. Iterative solution methods[7,8] and hybrid approaches [2] are possible candidates.

VI. PROBLEM SIZE LIMITATION DUE TO AVAILABLE SPACE

In order to evaluate the effect of computer resources on program capacity, storage space is first considered. Table 1 shows the limitations for a number of model types and different numbers of degrees of freedom -- if the solution is conducted in core, with 40,000 words available core storage. Both sparse representation, used in conjunction with an iterative solution method requiring no factorization of decomposition, and direct methods are considered. No account is taken of the storage overhead necessary to organize the data.

It is assumed that the total storage requirement remains practically the same regardless of the data handling method used.

Table 2 presents similar figures for an out-of-core solution, assuming one million words of disk space are reserved for the stiffness matrix.

In both cases, the high demand on computer storage is evident -- particularly in the case of the E2S3 models. One may notice, for example, that with an expansion of storage allocation from 40,000 to 1,000,000 words, the maximum E3S3 grid size, for a heat transfer problem, increases only from a (7x7x7) to a (14x14x14); i.e. double the grid resolution for a 25 fold increase in storage space.

The following is a key to the information contained in Tables 1 and 2:

d = (subscript) refers to direct method,

i = (subscript) refers to iterative method,

N_d, N_i = number of nodes per dimension (grid density),

f = number of degrees of freedom/node,

U_d, U_i = number of unknowns,

b = maximum half-band-width,

S_d, S_i = storage requirement for direct and iterative solutions,

N_d, N_i = program capacity for direct and iterative solutions, in terms of grid density resolution, and

σ = matrix sparsity factor (see below).

VII. GENERALIZATION OF THE BAND-WIDTH CONCEPT

The concept of the "band-width", B, and half-band-width, b, have been extremely useful in program design. In section VI, the concept was used as a basis for evaluating the relationship between storage space available and maximum problem size for a chosen set of highly regular grids, representative of typical classes of physical problems. Schemes based on sparse representation, however, such as described under IV.4 or the block-oriented hypermatrix scheme described under IV.5, do not operate on this principle. Such schemes take into consideration only those elements different from zero, in both storage and computation. It is necessary to define, in more precise terms, a new set of parameters to describe the sparsity of an arbitrary matrix. It is hoped that these new parameters will play a role in measuring the performance of solution schemes, as well as in the prediction of solution times, and node and

Model	U	b_o	b	$\sigma \simeq 2b/U$	$S_d = bU$	$S_i = b_o U$	f	N_d	U_d	N_i	U_i
E1-S1	$f \cdot N$	$2f$	$2f$	$4/N$	$2f^2N$	$2f^2N$	1	20,000	20,000	20,000	20,000
							3	2222	6666	2222	6666
E2-S2	$f \cdot N^2$	$5f$	$f(N+2)$	$\frac{2(N+2)}{N^2}$	$f^2N^2(N+2)$	$5f^2N^2$	1	33	1089	89	7921
							3	15	675	29	2523
E2-S3	$4fN(N-1)$	$8f$	$8f(N-1)$ *	$4/N$	$32f^2N(N-1)^2$	$32f^2N(N-1)^2$	1	11	440	35	4760
							3	5	240	12	1584
E3-S3	$f \cdot N^3$	$14f$	$f(N^2+N+2)$	$\frac{2(N^2+N+2)}{N^3}$	$f^2N^3(N^2+N+2)$	$14f^2N^3$	1	8	512	14	2744
							3	5	375	6	648

TABLE 1. Problem Size Limitation for In Core Solution

Available Core Space for Coefficient Matrix = 40,000 Words

*It is possible to reduce this to $2f(N+1)$, if a sparse scheme is utilized.

Model	U	b_o	b	$\sigma=$	S_d	S_i	f	N_d	U_d	N_i	U_i
E1-S1	$f \cdot N$	2f	2f	$4/N$	$2f^2N$	$2f^2N$	1	500,000	500,000	500,000	500,000
							3	55,555	166,665	55,555	166,665
E2-S2	$f \cdot N^2$	5f	f(N+2)	$\frac{2(N+2)}{N^2}$	$f^2N^2(N+2)$	$5f^2N^2$	1	99	9801	447	199,809
							3	47	6627	149	66,603
E2-S3	$4fN(N-1)$	8f	8f(N-1)	$4/N$	$32f^2N(N-1)^2$	$32f^2N(N-1)^2$	1	32	3968	177	124,608
							3	15	2520	59	41,064
E3-S3	$f \cdot N^3$	14f	$f(N^2+N+2)$	$\frac{2(N^2+N+2)}{N^3}$	$f^2N^3(N^2+N+2)$	$14f^2N^3$	1	15	3375	41	68,921
							3	10	3000	20	24,000

TABLE 2. Problem Size Limitations for Out of Core Solution

Available Disk Space = 10^6 Words

element renumbering, for the purpose of solution cost minimization.

VII.1 *Active Local Bandwidth*

This is defined as the total number of non-zero elements in a particular row or column. It is necessary to distinguish between the active local bandwidth before and after decomposition (or factorization). The former, which is smaller, is important in iterative schemes, while the latter determines the computational effort in a direct scheme. Symmetric coefficient matrices will now be considered, although a generalization to non-symmetric matrices is straightforward. We introduce the following symbols:

B_{o_i} Initial active local bandwidth for row (or column) i.

b_{o_i} Initial active local half bandwidth for row i. b_{o_i} is a count of the non-zero entries horizontally to the right, starting with the diagonal elements.

c_{o_i} Initial active local half bandwidth for column i. c_{o_i} is measured vertically up, starting with the diagonal.

B_i Active local bandwidth for row (column) i. It is equivalent to B_{o_i} for the coefficient matrix after factorization or decomposition.

b_i & c_i Active local half bandwidths for row and column i, respectively, after element propagation due to a direct solution.

The reader should notice that

$$B_{o_i} = b_{o_i} + c_{o_i} - 1 \tag{9}$$

and

$$B_i = b_i + c_i - 1 \tag{9a}$$

VII.2 *Matrix Profile*

This is defined as an array containing all local bandwidth values. It describes the matrix sparsity in a more accurate fashion than the usual (maximum) bandwidth concept.

$$\underline{B}_o = \{B_{o_1}\ B_{o_2} \ldots\ldots B_{o_i} \ldots\ldots B_{o_U}\} \quad (10)$$

is the initial matrix profile, and

$$\underline{B} = \{B_1\ B_2 \ldots\ldots B_i \ldots\ldots B_U\} \quad (11)$$

is the (final) matrix profile, after element propagation.

Similarly for initial row and column (half) bandwidth profiles we have

$$\underline{b}_o = \{b_{o_1}\ b_{o_2} \ldots b_{o_i} \ldots b_{o_U}\} \quad (12)$$

and

$$\underline{c}_o = \{c_{o_1}\ c_{o_2} \ldots c_{o_i} \ldots c_{o_U}\} \quad (13)$$

and for final profiles we define

$$\underline{b} = \{b_1\ b_2 \ldots b_i \ldots b_U\} \quad (14)$$

and

$$\underline{c} = \{c_1\ c_2 \ldots c_i \ldots c_U\} \quad (15)$$

VII.3 *Generalized Matrix Parameters*

The following parameters are introduced. They represent useful scalar measures of sparse matrix properties utilized to measure space and computational requirements. Again only symmetric matrices are considered, although the generalization to non-symmetric matrices is straightforward.

VII.3.1 *Mean Bandwidth*

The mean bandwidth of a matrix is defined as:

$$\bar{b} = \left(\sum_{i=1}^{U} b_i\right)/U. \tag{16}$$

This parameter is utilized primarily to measure storage requirements.

VII.3.2 *Matrix Sparsity Factor*

$$\sigma = 2\bar{b}/(U+1) \simeq 2\bar{b}/U \tag{17}$$

σ varies from $\frac{2}{(U+1)}$ for a diagonal matrix to 1 for a fully populated matrix.

VII.3.3 *Computational Bandwidth*

The number of floating point multiplications and additions involved in the triangularization of a matrix is approximately equal to $b^2U/2$, for a matrix with a constant local bandwidth. This leads to a bandwidth parameter, called the computational half bandwidth, which may be used for solution time estimates.

$$\hat{b} = \left[\sum_{i=1}^{U} b_i^2/U\right]^{\frac{1}{2}} \tag{18}$$

VIII. MEASUREMENT OF PROGRAM PERFORMANCE

Solution routines, based on a basic mathematical algorithm, are difficult to compare. The following performance measurement parameters are suggested to help comparison and evaluations.

VIII.1 *Storage Efficiency Parameters (E_s)*

The minimum storage required for a coefficient matrix is given by:

$$S_{min} = \bar{b}\, U. \tag{19}$$

However, due to program organization, additional space may be needed to structure the data. If the total amount of space required to store and manage the coefficient matrix is given by S, the storage

efficiency of the program is defined by:

$$E_s = \frac{S_{min}}{S}$$

VIII.2 *Computational Efficiency (E_c)*

The total amount of operations needed by the Gaussian elimination is approximately equal to

$$Q_G = \sum_{i=1}^{U} \left(\frac{b_i^2}{2} + 2B_i\rho \right) \qquad (20)$$

where ρ is the number of right-hand sides. However, considerable overhead is usually associated with data management. One way to measure efficiency is to compare the central processor time, T, taken by the program to solve the equations with the approximate time for the Gaussian elimination T_G, defined as

$$T_G = Q_G \, (t_M + t_A) \qquad (21)$$

where t_M and t_A are the CP times required to perform a floating point multiplication and an addition, respectively, on the given computer. The program computation efficiency, E_c, is defined as

$$E_c = \frac{Q_G \cdot (t_M + t_A)}{T} \qquad (22)$$

IX. COMPUTER RESOURCE REQUIREMENTS

A solution scheme has certain demands on the computer. In measuring the total burden on the system one has to consider the utilization of all its resources. In detail they are:

a. Central processor time to perform the necessary computations. Overhead is also present due to data management functions that the program has to perform.

b. Core space required to store program and data. In an out-of-core

solution, a part of this space is used as a dynamic working space containing, at any one time, only a fraction of the data required for the solution of the problem. Core space is an important and limited resource. This fact is reflected in most charge algorithms, which use the amount of core utilized by a program as weighting factor in assessing system charges.

c. Input/Output (I/O) time. Ignoring hard to measure factors such as system I/O buffering and virtual memory machine operation, each explicit request by the program to the system to either read or write a block of data from an I/O device, typically a disk, constitutes a demand on system resources. The system I/O charges are based on both the initial disk access (es) and the amount of data transferred. Both elements will be used as criterion in evaluating the I/O requirements of a particular algorithm. It is interesting to note that in large computer systems I/O operations require extensive amounts of associated CP.

d. Disk Space Requirements. Finally, a certain amount of disk space is required to store primary problem data as well as organizational data used to manage it. Disk requirements may or may not be a factor in assessing computer run charges. If the data is retained on disk permanent files after job completion, disk storage is charged.

IX.1 *Comparing Resource Requirements for Different Solution Schemes*

In order to determine the suitability of the various schemes for the solution of typical model types, we have to assess the total effect due to all four factors listed above. Studies by the authors and others (see for example [9]), show that central processor time utilized in the basic solution steps is more or less invariable -- regardless of the mathematical tool and data handling scheme used. It can be also shown that disk space requirement is practically constant. It follows that the two most important factors are those of core storage and the I/O time. The purpose of this section is to study the amount of both resources and their dependence on the model type and the data handling scheme. Only the fundamental data schemes will be considered. The following gives a brief description of each method. Each scheme, except for IB, is designated by a two letter code. The first denotes the storage unit which may be a

row (R), a group of rows (G) or a sub-matrix (S). The second denotes the solution (pivoting) unit. The method may solve row by row or group by group.

a. In-core banded solution (IB).

b. Out-of-core banded solution (RR), based on active bands. Each single row is stored as an independent record and may be read or written with a single I/O instruction. Pivoting is by rows.

c. Out-of-core banded solution. Rows stored in blocks. Solution proceeds row by row (GR). For a particular problem, the more rows there are in a group, the more core is needed and the less the number of I/O operations. In balancing the two factors, program performance may be optimized as shown in section VIII.

d. Out-of-core banded solution. Rows stored in blocks. Pivoting is by groups (GG). This method is the most economical in I/O operations (see Table 3), but requires excessive core storage for large problems.

e. Out-of-core hypermatrix solution (SG) in which the grouping of unknowns is done on both rows and columns in a compatible manner. The coefficient matrix is divided into sub-matrices. Each sub-matrix is read into core or written on disk with one I/O instruction. This method is the most economical in core storage utilization for large problems.

The core storage requirement for the hypermatrix is that sufficient to hold four or five sub-matrices, as well as pointers to a number of related sub-matrices. In the simplest possible of schemes the full pointer matrix is stored in core. This approach, however, severely limits problem size. In order to reduce the amount of pointer storage, any of the approaches applied to the coefficient matrix itself may again be utilized. In particular, the following possibilities exist:

(i) Pointer matrix stored in core, as a full matrix (SG-IF).

(ii) Pointer matrix stored in core, as a banded (sparse) matrix (SG-IB).

(iii) Pointer matrix stored out of core, as a banded matrix, only the relevant rows are held in core (SG-OB). We will only consider the case where each record stores the addresses to a row of sub-matrices, in a sparse format. All active address rows are pre-

sent in core.

(iv) Pointer matrix stored out of core, as a hypermatrix (SG-H).

The SG-OB method for storage of pointer matrix rows will be considered in assessing core storage. It will be assumed that the additional I/O operations to handle pointers have a negligible effect. It is also assumed that, for the hypermatrix scheme, the right-hand side will be stored independently of the coefficient matrix, in compatible blocks. It is possible, therefore, to solve for many right-hand sides simultaneously. If the number of right-hand sides exceeds the number of rows allowable in a partition, the right-hand side matrix can be partitioned column-wise also. This case will not be considered here.

IX.2 *Comparison of Computer Resource Requirements*

Table 3 shows a comparison between the various solution algorithm/data handling combinations and their core and I/O requirements. A banded matrix is assumed. In interpreting the given data, however, one must realize that a well written sparse scheme will essentially provide the same results, with the active band width taking the place of the maximum band width. Each method is designated with the letter identifier used in the last section (IB, RR, GR, GG, SG). In addition, the letter T will denote the triangularization (Gaussian Elimination) scheme and the letter F the factorization (Cholesky, Crout, etc.).

In the banded, row oriented solution schemes, it is assumed that the right-hand side(s) are stored with the corresponding rows. In the hypermatrix scheme the right-hand sides are stored in separate blocks. The following notation is employed:

r = No. of rows in a block, or submatrix size.

s = No. of non-zero submatrices in a hyperrow $\simeq$ b/r

G = Total no. of partitions $\simeq$ U/r.

Data Handling Method	Core Requirement	Number of I/O Calls	Record Size
IB-T IB-F	Ub	0	-
RR-T	2b	4bU	b
RR-F	2b	5bU	b
GR-T	b(r+1)	4sU	br
GR-F	b(r+1)	5sU	br
GG-T	2br	4sG	br
GG-F	2br	5sG	br
SG-T	$4r^2 + s^2$	$\frac{G}{2}[3s^2 + 13s - 8]$	r^2
SG-F	$5r^2 + s^2$	$\frac{G}{2}[3s^2 + 11s - 6]$	r^2

TABLE 3. Comparison of Core Requirements and I/O Operations for Data Handling Schemes

IX.3 *Examples*

Figures 2, 3 and 4 describe program requirements during the solution of a two-dimensional model (E2S2) with two degrees of freedom per node. In solution schemes based on row and/or column grouping, blocks of 30 rows each are assumed. Fig. 2 shows core requirements for the various methods as a function of problem size. The group storage, group reduction scheme, GG, shows an advantage over the hypermatrix scheme, for models up to N=28 (U≃1600). Figures 3 and 4 shows the number of I/O calls, and demonstrates definite superiority of GG and SG over the GR and RR schemes. Figure 4 shows a slight advantage for the SG over the GG method.

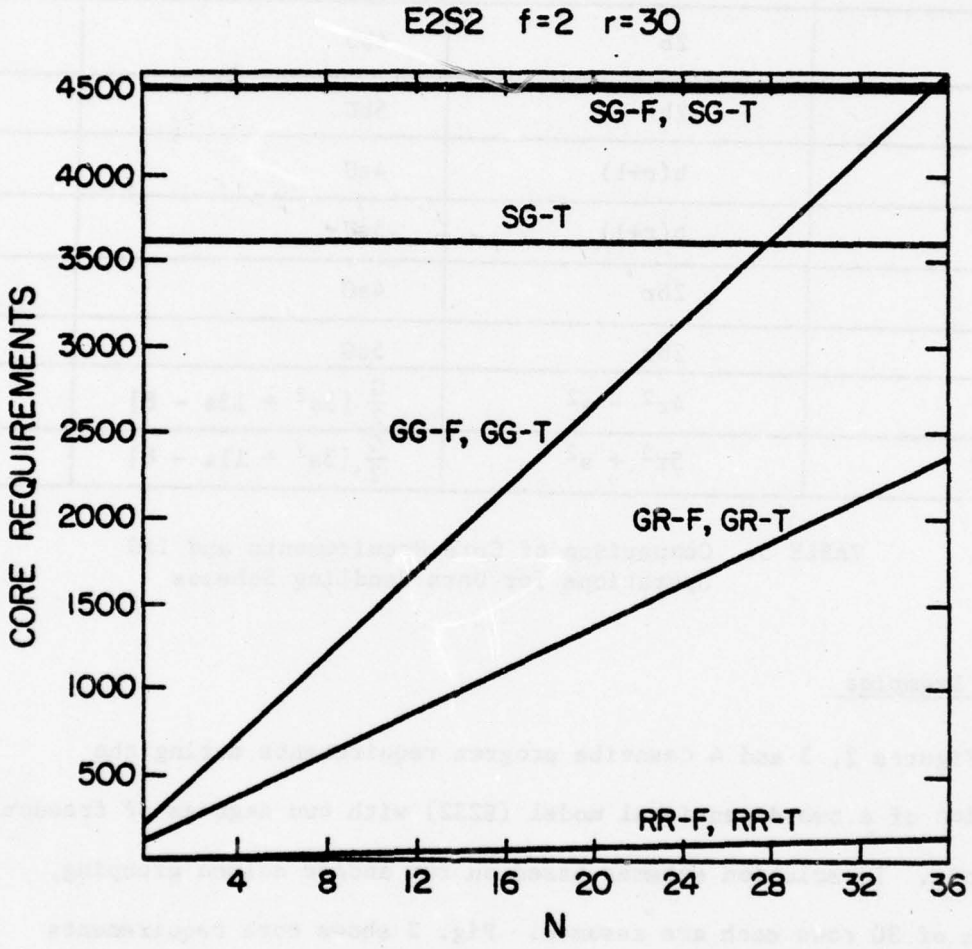


Figure 2. Core Requirement for a 2-D Model, as a Function of Problem Size

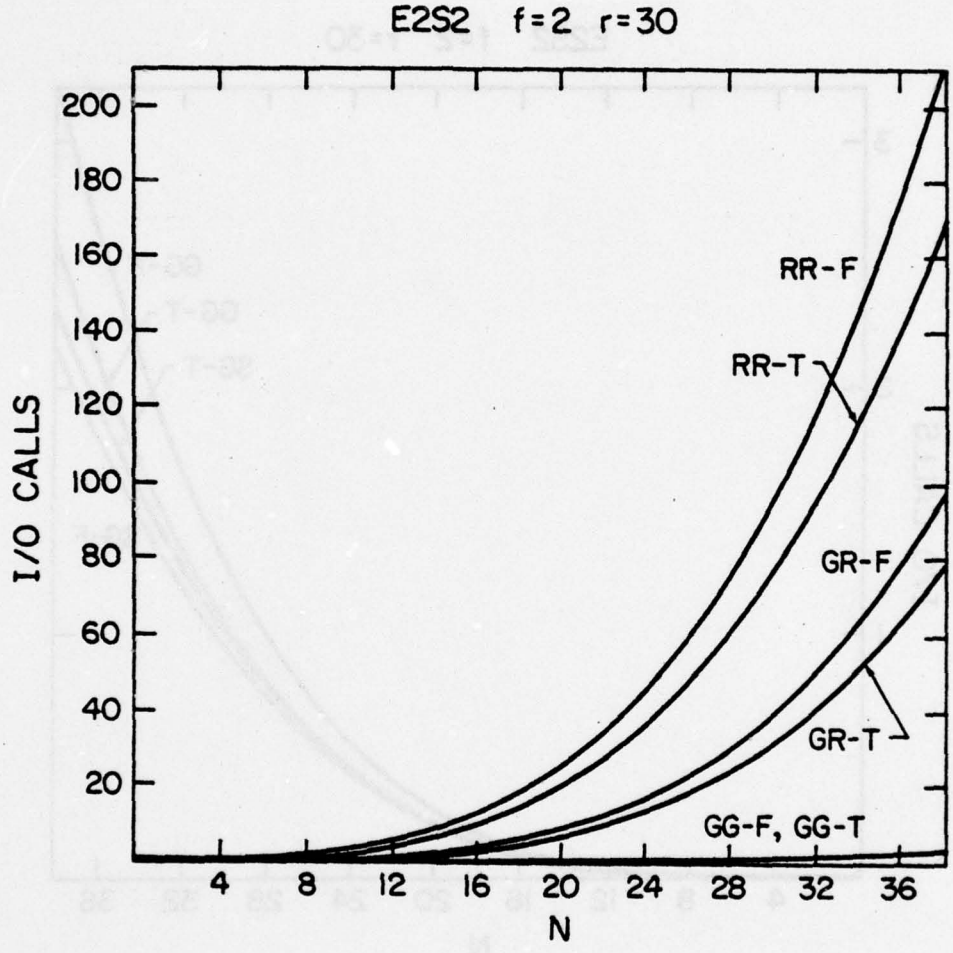


Figure 3. I/O Operation for a 2-D Problem as a Function of Problem Size

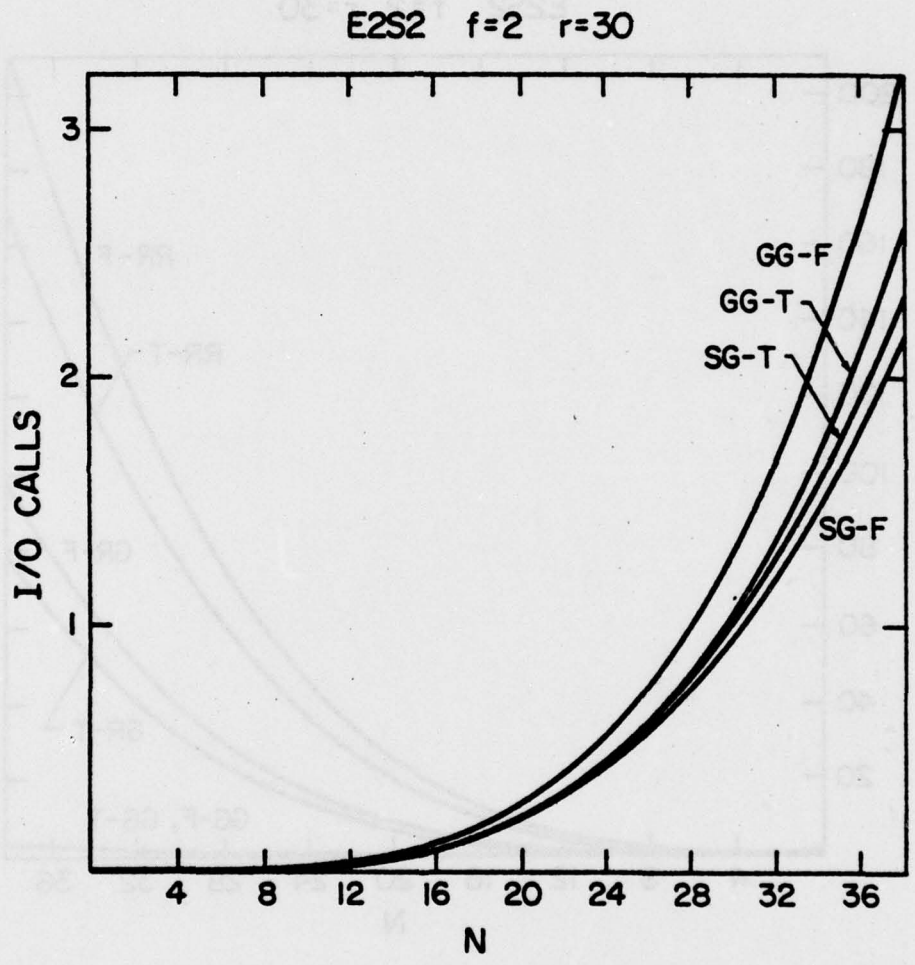


Figure 4. comparison of I/O Times (only GG, SG) for an E2S2 Problem

Figures 5, 6, and 7 show the core and I/O requirements for a two-dimensional problem with 3200 unknowns and a half bandwidth of 84 as a function of partition size. As seen from figure 4, the hypermatrix scheme uses somewhat less core than GG for partition sizes of up to 35 or 40. GG and SG require less I/O than GR throughout, with SG again holding a definite edge.

It appears, therefore, from figures 2 to 7 that both the GG and SG schemes are superior in handling of two-dimensional problems.

Moving to shell-type problems (E2S3), figure 8 illustrates core requirements as a function of partition size. It is quite apparent that, for large bandwidth problems, the hypermatrix scheme requires substantially less core. From an I/O point of view (fig. 9), GG is again inferior to SG.

The same conclusions are valid for solids problems, as illustrated by figures 10 and 11.

X. CHOICE OF OPTIMUM PARTITON SIZES

In solution algorithms involving matrix partitioning in one direction (GR-T, GR-F, GG-T, GG-F) or in both directions (SG-T and SG-F), large block sizes result in efficient disk operation at the expense of core space. The optimum partition size depends on the hardware available and the charging algorithm utilized by the computer center. In order to illustrate how one may optimize the size of the partition for a particular system, two cases are considered. One case represents a typical computer center environment where central processor time, I/O time and core requirements are all taken into account in the charge algorithm. The other case is that of a mini-computer in which residence

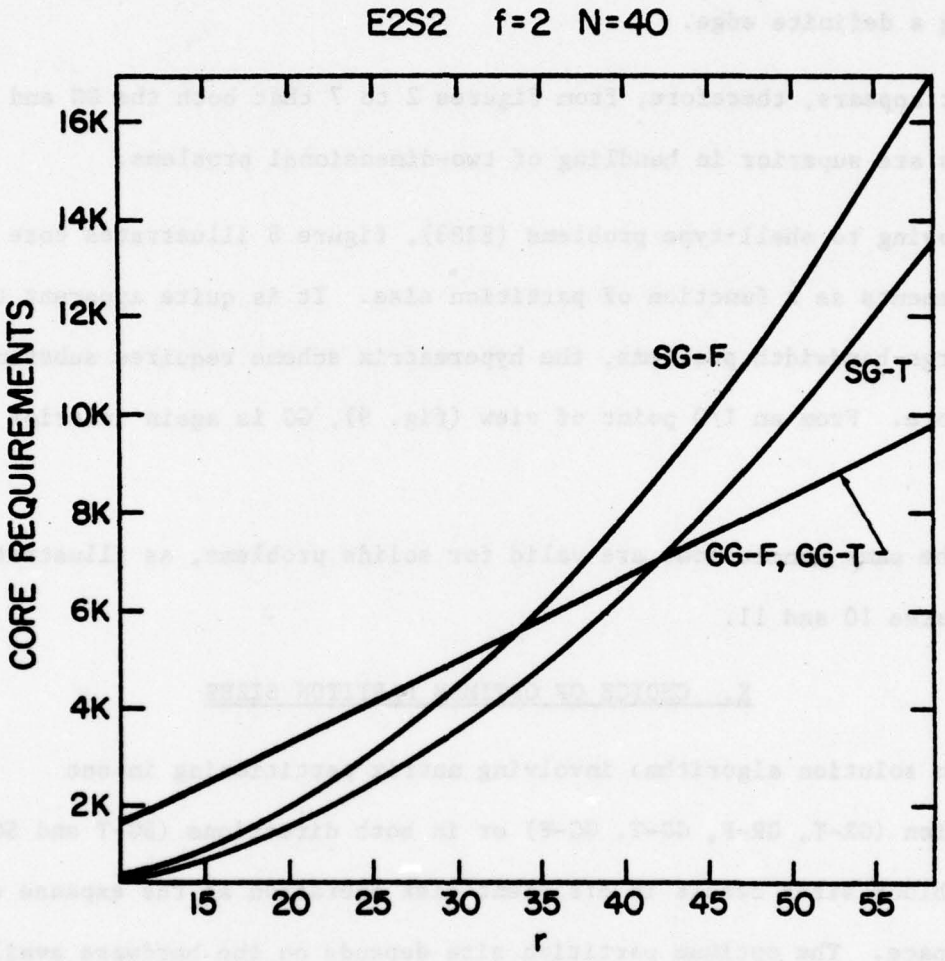


Figure 5. Core Requirement for an E2S2 Model as a Function of Partition Size (U=3200, b+84)

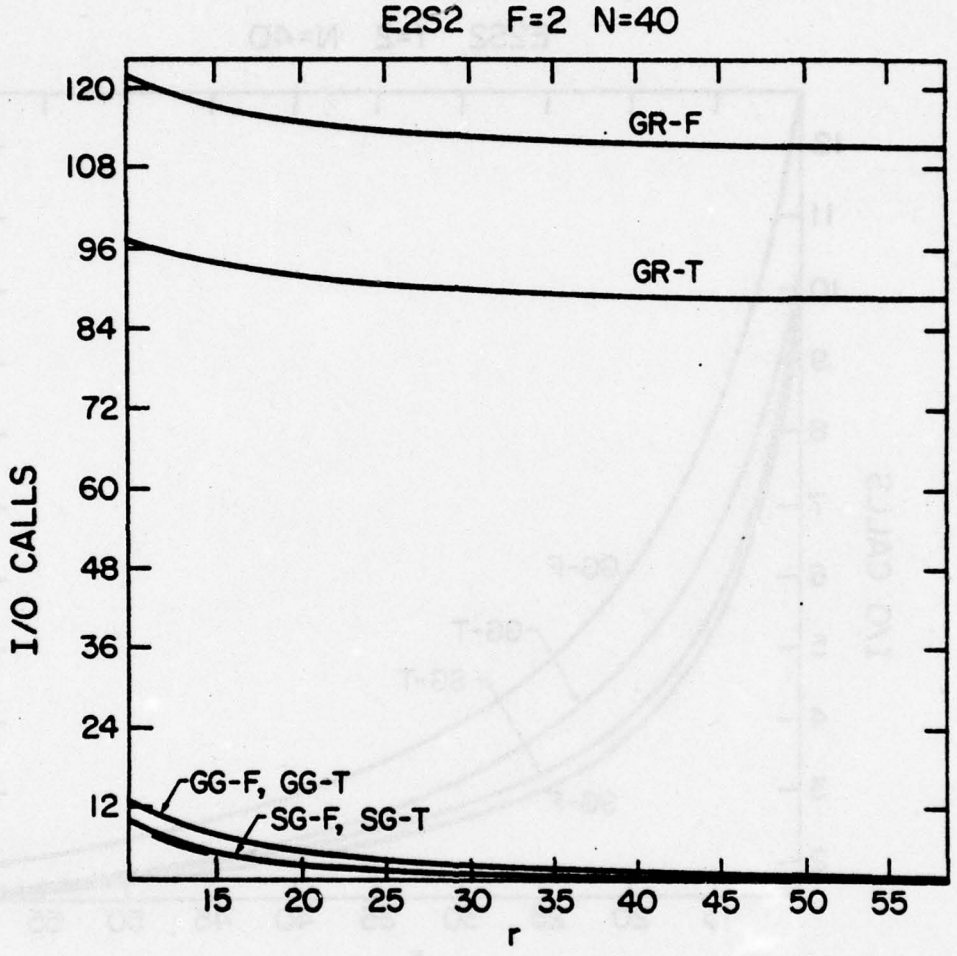


Figure 6. No. of I/O Calls for an E2S2 Problem as a Function of Partition Size (U=3200, b=84)

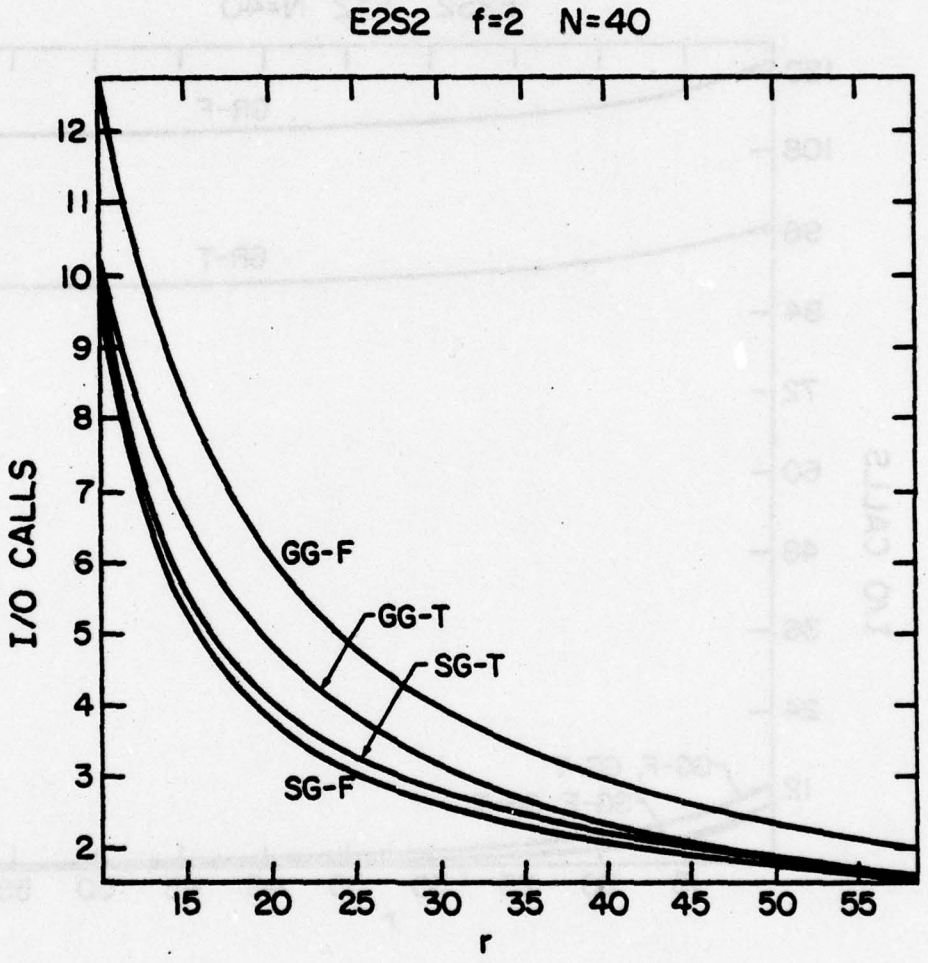


Figure 7. I/O Calls for Group Reduction Schemes (GG,SG) (U=3200, b=84)

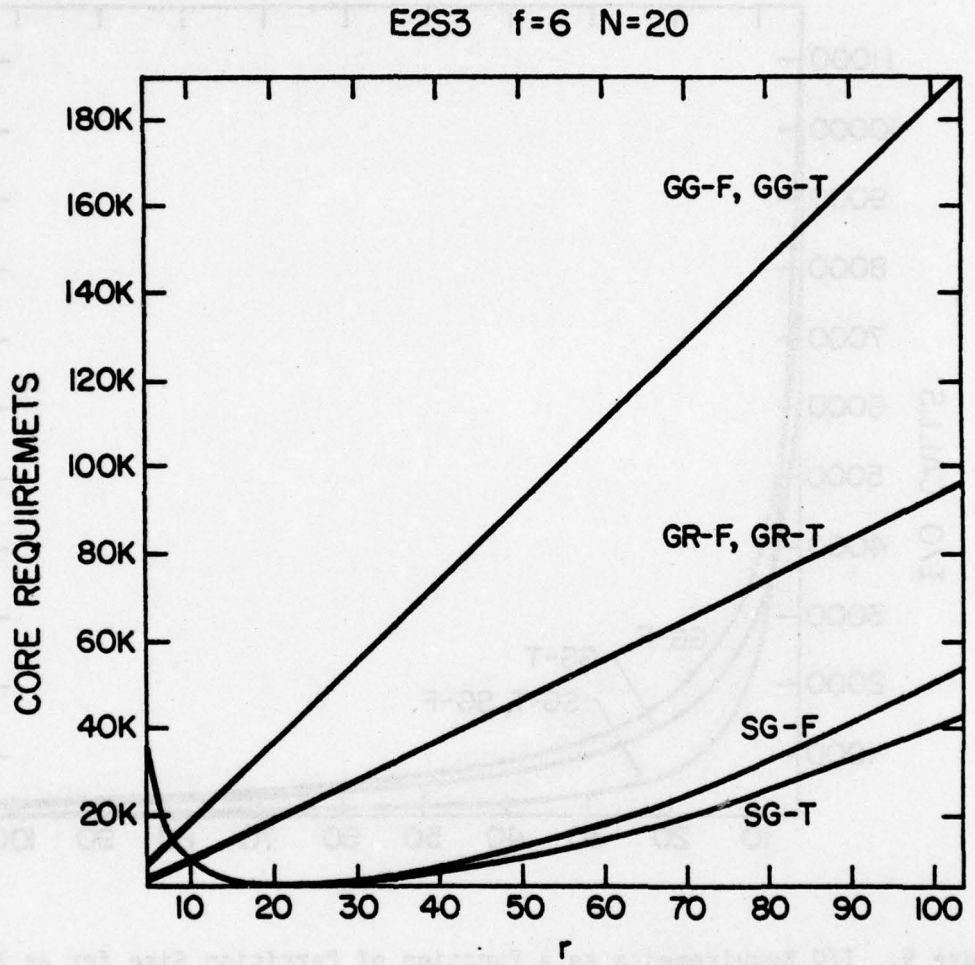


Figure 8. Core Requirements as a Function of Partition Size for an E2S3 Model (U=9120, b=912)

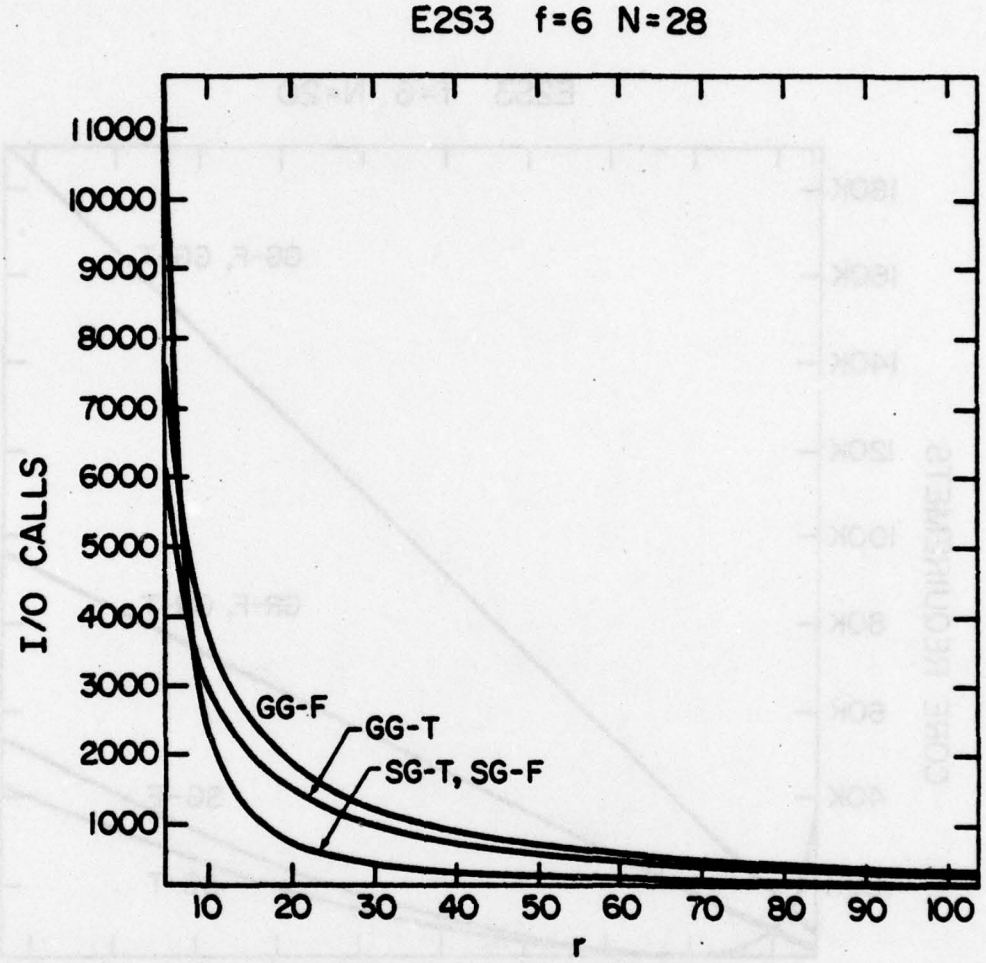


Figure 9. I/O Requirements as a Function of Partition Size for an E2S3 Model

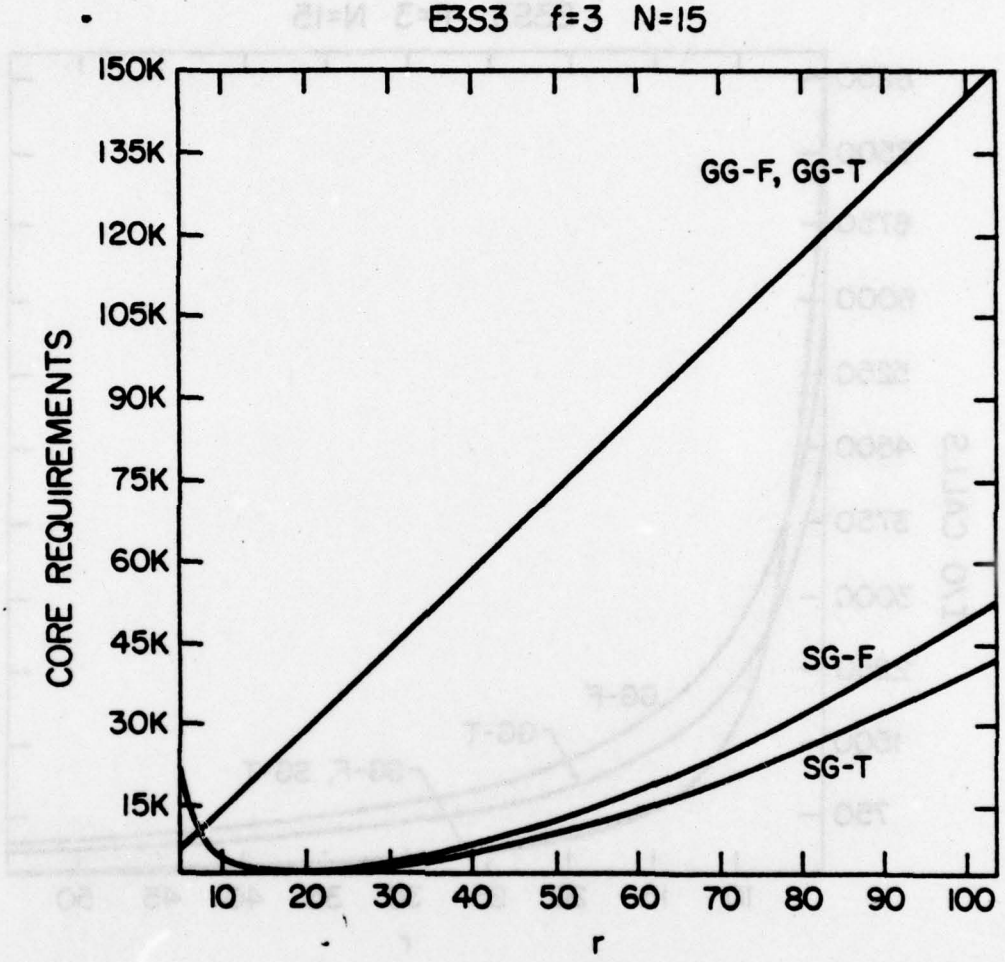


Figure 10 Core Requirements for a Solids Problem (U=10125, b=726)

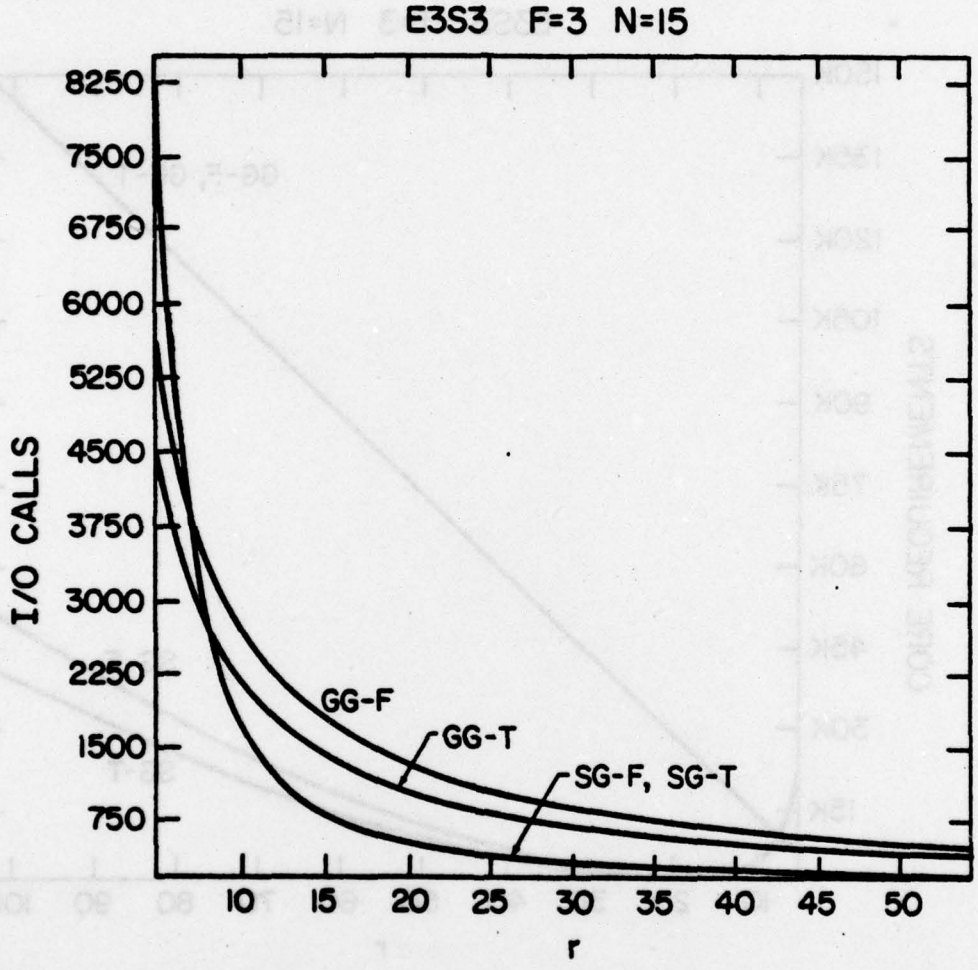


Figure 11. I/O Requirements for a Solids Problem (U=10125, b=726)

time and limited core are the only considerations.

X.1 *Computer Center Environment*

A typical charge algorithm in a computer center environment is given by the following formula**:

$$\text{Cost} = R * (\alpha C + \beta P + \gamma S * L)$$

where,

R = System cost/hour

C = Central processor time, in hours

P = Peripheral processor (I/O) time in hours. It is defined as the total time the program occupies an I/O channel, and is a function of both the number of disk accesses and amount of words transferred.

S = Core storage, in words

L = Larger of C and P.

Typical values for the constants are

R = $500

$\alpha = 0.6$

$\beta = 0.15$

$\gamma = 1.63 \times 10^{-5}$.

Furthermore, P is computed from

$$P = p_W W + p_A A$$

where

W = the number of words transferred,

and

A = number of accesses.

**Based on the University of Arizona Computer Center's CDC 6400 system

For the installation under consideration, the constants are given by

$$p_w = 1.11 \times 10^{-8} \text{ hours/word}$$

$$p_A = 1.04 \times 10^{-6} \text{ hours/access.}$$

Assuming the program has a computational efficiency of E_c, the central processor time is given by:

$$C = \frac{Q_G(t_m + t_A)}{E_c} = \frac{(t_m + t_A)}{E_c} \left(\frac{Ub^2}{2} + 4b\rho\right).$$

The peripheral processor (I/O) time for the various methods is given by Table 3. M can be put in the form

$$M = M_S + M_P + M_D$$

where

M_S is system memory requirement

M_P is the user program requirement, excluding data,

and

M_D is the data requirement, given in Table 3 for the different schemes.

Case Study 1

Let us examine an E2-S3 model with

$$N = 20$$

and

$$f = 6$$

$$U = 4f\,N(N-1) = 9120$$

and

$$b = 8f(N-1) = 912.$$

Let us assume 5 loading cases. Furthermore, for a CDC 6400,

$$t_m = 57 \times 10^{-7} \text{ sec.}$$

$$t_A = 11 \times 10^{-7} \text{ sec.}$$

We assume that, for a typical program,

$$E_c = 0.8$$

also

$$M_S = 5000 \text{ words}$$

$$M_P = 12000 \text{ words.}$$

Figure 12 shows the computed cost for the given problem, on the chosen installation using both the GG and SG schemes. It is interesting to note the sharp minimum exhibited by the GG method at about r = 35 to 40, due to the central processor time becoming predominant, coupled with the high core requirements. One notes further that the required core reaches 64K at r = 26 and 131K for r = 62. On the other hand, the hypermatrix scheme (SG) exhibits a flat minimum extending from r = 20 to r = 40. The minimum cost for the SG is approximately $5,500, compared to $10,300 for GG. These facts show not only the clear superiority of the hypermatrix scheme for complex shell problems, but, assuming the charge algorithm is reasonable also the unsuitability of the computer installation for handling problems of this type and magnitude.

XI. MINI-COMPUTER ENVIRONMENT

In a mini-computer environment, the cost is a function of the residence time rather than system time. A typical cost formula** will

---

**Based on a PDP-15 computer, 32K of core, 18 bit words.

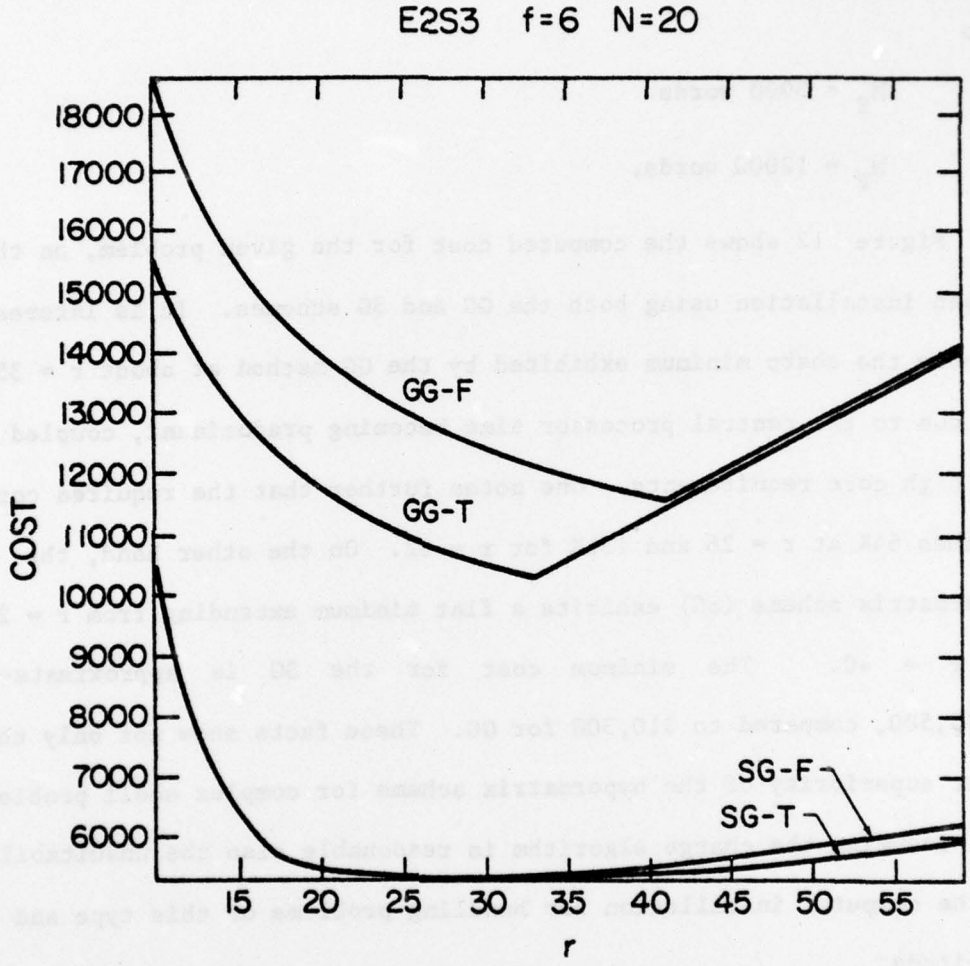


Figure 12. Effect of Partition Size on Solution Costs for Row Blocking (GG) and the Hypermatrix Scheme (SG) (U=9120, b=912)

be of the form:

$$\text{Cost} = R * (C + \beta P).$$

Typical values are

$$R = \$15.00/\text{hour}$$

and

$$\beta = 8.333 \text{ E-6 hours/call}$$

$$C = \frac{(t_m + t_A)}{E_c} \left(\frac{Ub^2}{2} + 4b\rho\right)$$

where, for a mini-computer without floating point processor

$$t_A = 5.556 \times 10^{-8} \text{ hours}$$

and

$$t_m = 5.833 \times 10^{-8} \text{ hours.}$$

With a floating point processor,

$$t_m = 4.167\text{E-}9$$

$$t_A = 2.778\text{E-}9$$

$$P = \text{No. of I/O calls}$$

The number of rows per group is not determined by the charge algorithm, but rather by the availability of core. Assuming a core of 32K 18 bit words, and assuming that the system occupies 6K words, and the program 5K, then a total of 21K is available for data. Assuming extended accuracy (3 18 bit words per floating point number) then, for GG,

$$21000 = 2br \times 3 = 6br.$$

XI.1 *Case Studies*

The mini-computer cost algorithm is applied to the same complex shell problem used previously, with 9120 degrees of freedom and a half bandwidth of 912, (see figures 13 and 14). Since the amount of core available is limited to 32K of 18 bit words, the curves are discontinued when this limit is reached. It is obvious that the cost is reasonable if a floating point processor is available -- $600 for SG and $1100 for GR. The running times, however, are 40 and 70 hours, respectively! It would appear that improvement of mini-computer speeds is required before problems of this magnitude can be tackled in one computation.

XII. CONCLUSION

The paper establishes a systematic approach to the comparison of basic solution techniques and data handling strategies. New parameters are introduced, which may be used to describe sparsely populated matrices, and estimate times and costs of solving the equations, in advance of the computation. It is demonstated how one can optimize run costs for a particular solution algorithm, a particular problem and a particular installation. From the results it appears that both a partitioned banded approach and the hypermatrix scheme are suitable for the solution of medium size problems with a narrow active bandwidth. For large shell and solids problems, the hypermatrix scheme seems to be more efficient. Mini-computers compare favorably with large systems in running costs, but may require unacceptable running times for large problems.

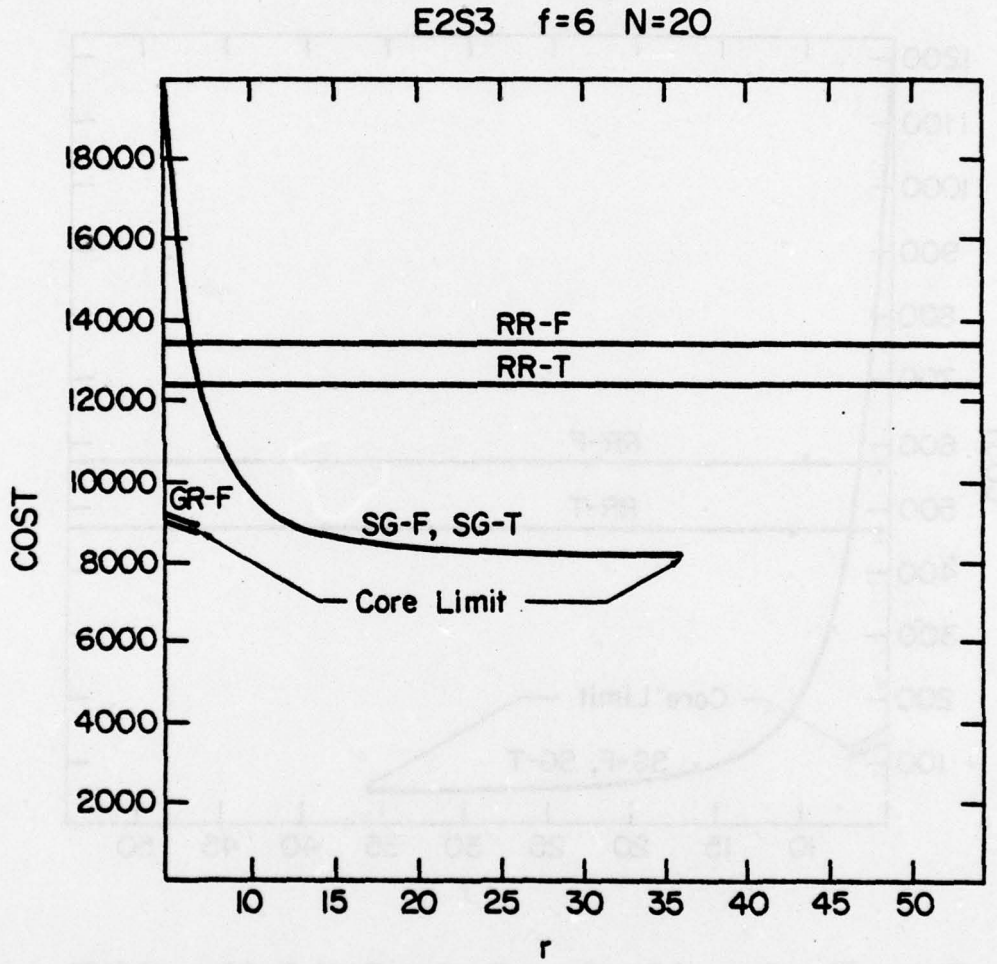


Figure 13. Cost of Run for a Complex Shell Problem (U=9120, b=912) On a Mini-Computer Without a Floating Point Processor

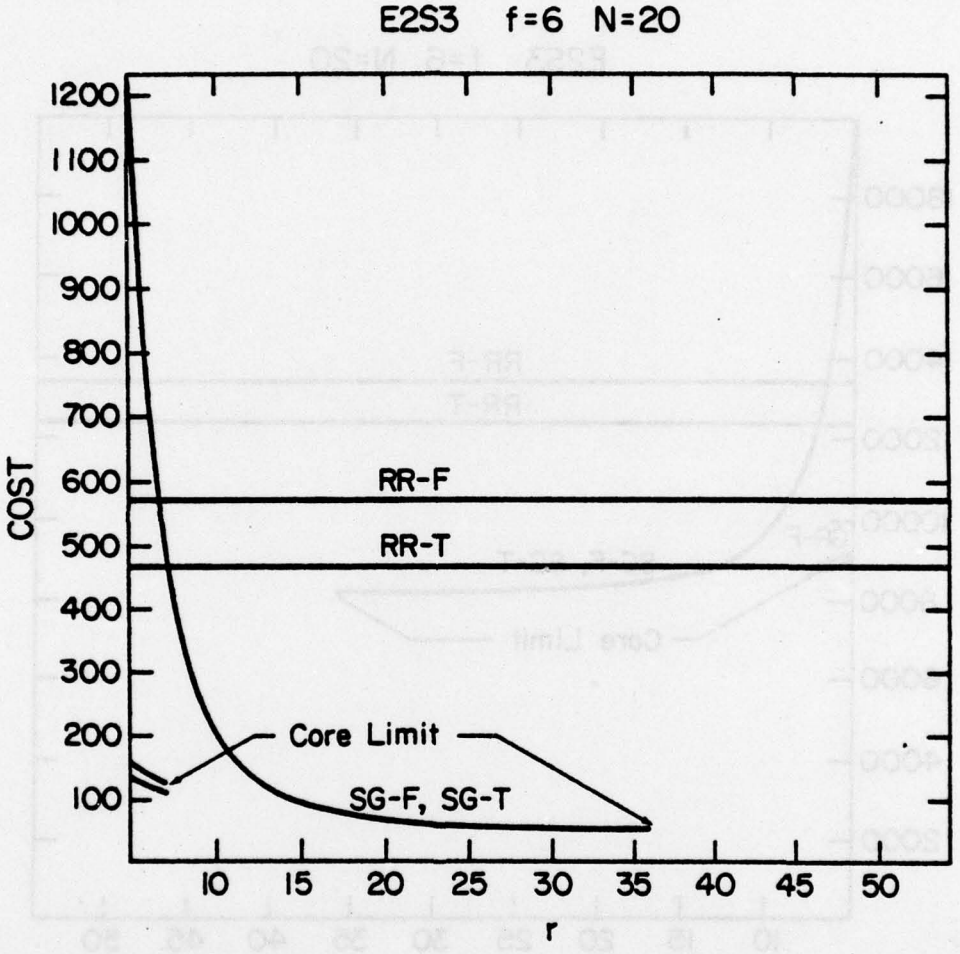


Figure 14. Cost of Run for a Complex Shell Problem (U=9120, b=912), For a Mini-Computer with a Floating Processor

ACKNOWLEDGMENT

The research has been supported by the Office of Naval Research under contract no. ~~N00014-67-A-0709-0016~~ and the American Bureau of Shipping.

The manuscript was typed by Ms Juanita Alvarez and drawings by Lyn Davis.

This paper is dedicated to J.H. Argyris on the occasion of his 65th birthday.

REFERENCES

1. Wilkinson, J.H., *The Algebraic Eigenvalue Problem,* Oxford University Press, Oxford, (1965).

2. Rashid, Y.R., High Speed Computing of Elastic Structures, Vol. 1 & 2, *Proc. of the symposium of International Union of Theoretical and Applied Mechanics,* (Aug. 1970).

3. Kamel, H.A., Liu, D., White, E.I., The Computer in Ship Structure Design, paper presented at the *ONR International Symposium on Numerical and Computer Methods in Structural Mechanics,* University of Illinois, Champaign, Illinois, (Sept. 8-10, 1971).

4. Phansalkar, S.R., Solution of Large Systems of Linear Simultaneous Equations by Inverse Decomposition, *Computers and Structures,*Vol. 5, Nos. 2/3, pp. 131-144, (June 1975).

5. Schwarz, R.H., Rutishauser, H. & Stiefel, E., Numerical Analysis of Symmetric Matrices, *Prentice Hall Series in Automatic Computation,* (1973).

6. Stiefel, E.L., *An Introduction to Numerical Mathematics,* Academic Press Inc., (1963).

7. Hestenes, M.R. and Stiefel, E., Method of Conjugate Gradients for Solving Linear Systems, *Journal of Research of the National Bureau of Standards,* Vol. 49, No. 6, pp. 409-436, (1952).

8. Fried, I., A Gradient Computational Procedure for the Solution of Large Problems Arising from the Finite Element Discretization Method, *International Journal for Numerical Methods in Engineering,* Vol. 2, No. 4, pp. 477-494, (1970).

9. Cassell, A.C. and Hobbs, R.E., Dynamic Relaxation in High Speed Computing of Elastic Structures, *Proc. of the Symposium of International Union of Theoretical and Applied Mechanics,* (Aug. 1970).

10. Wilson, E.L., SAP -- A General Structural Analysis Program for Linear Systems, paper presented at the *2nd U.S.-Japan Seminar on Matrix Methods of Structural Analysis and Design,* (Aug. 1972).

11. McCormick, C.W., The NASTRAN Program for Structural Analysis, paper presented at the *2nd U.S.-Japan Seminar on Matrix Methods of Structural Analysis and Design,* (Aug. 1972).

12 Kamel, H.A., Lamber, R.L., Solution of Structural Eigenvalue Problems Using Sparsely Populated Matrices, paper presented at a conference on *Computer Oriented Analysis of Shell Structures,* Palo Alto Research Laboratory, Palo Alto, California, (Aug. 10-14, 1970).

13 Argyris, J.H., Kamel, H.A., et. al., *ASKA an Automatic System for Kinematic Analysis,* Programmer's Manual, (Oct. 1965).

14. Kamel, H.A., *DAISY User's Manual,* Internal Report, (May 1968).

15. Kamel, H.A., McCabe, M.W., The GIFTS System, paper presented to the *International Symposium on Structural Mechanics Software,* University of Maryland, College Park, Maryland, (June 4-6, 1974).

16. Kamel, H.A., et al., *Annual Report to the Office of Naval Research,* under contract no. 5015-2212-50, (Feb. 1975).

17. Von Fuchs, G., Roy, J.R. and Schrem, E., Hypermatrix Solution of Large Sets of Symmetric Positive-Definite Linear Equations, *Comp. Method Appl. Mech. Eng. 1,* pp. 197-216, (1972).

18. Irons, B.M., A Frontal Solution Program for Finite Element Analysis, *Int. J. Num. Meth. Eng. 2,* pp. 5-32, (1970).

19. Melosh, R.J. and Banford, R.M., Efficient Solution of Load Deflection Equations, *J. Structural Div.,* ASCE, Vol. 95, No. ST4, pp. 661-676, (1969).